

Tworzenie własnej powłoki

Zamknięci w muszli

W tym miesiącu Steven Goodwin omawia sposób stworzenia własnego interpretera poleceń w prosty i bezbolesny sposób. Stworzymy sami powłokę systemową! **STEVEN GOODWIN**

Dla obserwatora z zewnątrz Linux jest jednorodnym systemem. Dla pozostałych jest to zgrana mieszanka jądra systemu, modułów i narzędzi, z których każde zajmuje się dokładnie pojedynczym zadaniem. Dotyczy to także procedury logowania do systemu. Znak zachęty do logowania (wymagający podania nazwy użytkownika i hasła) obsługiwany jest przez średnio przyjazny znak zachęty poleceń (umożliwiający uruchamianie programów). Ten znak zachęty nazywamy powłoką (lub interpreterem poleceń). A ponieważ jest to tylko kolejny program, więc możemy go zmodyfikować, dodać potrzebne funkcje lub nawet zastąpić go własnym rozwiązaniem. I właśnie to zamierzamy zrobić!

Walka z wiatrakami

Powłoka to interpreter poleceń, umożliwiający użytkownikom wprowadzanie komend, które w rezultacie wykona system. Bbrzmi to banalnie. Wpisujemy polecenie, powłoka je wykonuje. System operacyjny zajmuje się bardziej szczegółową stroną, sprawdzając uprawnienia plików, ładując plik, aktualizując tabelę procesów i sterując bitem SUID. Z kolei powłoka jest odpowiedzialna za przeadresowania, parametry symboli wieloznacznych (ang. wildcards), sterowanie potokami i wiele, wiele innych. Krótki opis umieszczono w Ramce 1.

Jako że powłokę można w każdej chwili zastąpić inną, wielu programistów starało się podmienić ją swoją własną wersją. Każda powłoka, tak jak każda z dystrybucji Linuksa, przeznaczona jest do określonych zadań. Niektóre (np. ash) są bardzo małe i oferują minimalny zestaw funkcji, a ich przeznaczeniem jest ochrona dysków. Inne (np. csh lub ksh) umożliwiają przeprowadzanie bardzo skomplikowanych operacji na skryptach.

Najbardziej popularną powłoką jest powłoka Linuksa o nazwie bash (skrót od ang. Bourne Again SHell), napisana przez Stephena Bournea. Ze względu na swoją wszechstronność powłoka ta stała się de facto standardem w Linuksie. My postaramy się wypełnić niszę, oferując powłokę posiadającą funkcje sterowania multimediami. A więc do dzieła! Rozpoczynamy prace nad *mmshell*!

Gdybym miał młotek...

mmshell ma posiadać podstawową funkcjonalność multimedialną (odtwarzanie plików MP3, sterowanie głośnością, dostęp do napędu CD) oraz umożliwiać korzystania z narzędzi zewnętrznych, opisanych w Tabeli 1: Narzędzia. Przy tak ograniczonym zestawie funkcji nie będziemy potrzebowali zaawansowanych metod edycji, czy bufora historii poleceń. Użyjemy zatem prostego interfejsu w stylu menu, dzięki



Esther Keller, visipix.com

czemu użytkownik nie będzie musiał wpisywać poleceń, ani pamiętać określonych przełączników poleceń. *mmshell* zaprezentujemy na przykładzie określonego użytkownika multimedialnego o nazwie *music*. Powinniśmy rozpocząć od opracowania tradycyjnego systemu menu. Następnie przeanalizujemy różnice pomiędzy zwykłymi programami a programami powłoki oraz postaramy się poradzić sobie z nimi. Na koniec zajmiemy się odpowiednim sposobem instalacji i użytkowania naszej nowej powłoki.

Przejażdżka

Większość programistów napisała przynajmniej raz w swoim życiu system menu. Nie są one ani zbyt wielkie, ani zbyt wyrafinowane. Naprawdę! Nasz samodzielny program menu może wyglądać tak, jak pokazano na Listingu 1.

Ramka 1. Niektóre cechy powłoki

Zarządzanie zmiennymi środowiskowymi, np. PS1 i PATH. Druga z nich to uporządkowana lista katalogów, którą przegląda powłoka w poszukiwaniu programów.

Polecenia wewnętrzne. Niektóre polecenia, jak np. *cd*, *export* czy *history*, zostały umieszczone wewnątrz kodu samej powłoki. Zwiększa to prędkość działania systemu (dostęp do dysku nie jest potrzebny).

Rozszerzenia symboli wieloznacznych. Umożliwiają np. zmianę nazwy pliku *rm*

**.jpg* na *rm file1.jpg file2.jpg file3.jpg*.

Wbudowana funkcja edytowania. Dotyczy wypełniania pól, skrótów klawiaturowych (np. CTRL+A lub CTRL+E) umożliwiających powrót do początku lub końca bieżącej linii.

Przeadresowanie i potoki. Każdy z trzech podstawowych strumieni danych (wejściowy, wyjściowy, błądów) może być skierowany w kierunku z lub do plików przy pomocy potoków.

Tabela 1. Narzędzia

<i>mplayer</i>	Znakomite narzędzie do odtwarzania plików audio i wideo, obsługujące kilkanaście kodeków, także systemu Windows. Jeżeli nie potrzebujemy zaawansowanej obsługi wielu kodeków, odpowiedniejszym programem będzie <i>mpg123</i> (<i>mplayer</i> może powodować problemy wymagające wykonania reset terminala).
<i>cdcd</i>	Odtwarzacz CD pracujący z wiersza poleceń. Bardzo użyteczny. Obsługuje także bazę danych CD i <i>cddb</i> . Jeżeli program nie odnajdzie pliku konfiguracyjnego (<i>.cdcdrc</i>) w katalogu głównym użytkownika, uruchomiony zostanie tryb interaktywny w celu uzyskania odpowiednich danych od użytkownika.
<i>aumix</i>	Odchudzony Mikser posiadający tylko podstawową funkcjonalność.

Patrząc na Listing 1 z perspektywy programisty zauważymy, że jest to bardzo prosty program. Być może będziemy musieli zmienić przełączniki urządzeń dla programów *cdcd* i *aumix*, ale wymagają one co najwyżej podstawowych wyjaśnień. Prawdopodobnie nie trzeba nawet dodawać opcji dla kompilatora, zrobimy to jednak:

```
gcc mmshell.c -o mmshell
```

Pamiętajmy jednak o tym, że program ten będzie wykorzystywany jako powłoka, musimy zatem rozważyć także inne kwestie z tym związane.

Wody rzeki Orinoco

Bez względu na to, czy będzie to interpreter poleceń czy też samodzielny program, musimy przede wszystkim pomyśleć nad podstawową strukturą programu. W końcu powłoka jest po prostu programem, więc niczego innego nie

powinniśmy się spodziewać. Program uruchamia się standardowo dzięki funkcji *głównej*, wraz z nazwą pliku wykonywalnego *argv* [0] (poprzedzonym znakiem minus, gdy jest uruchamiany jako powłoka) i kończy pracę (wylogowując użytkownika z danego procesu) po zakończeniu funkcji *main*. Powłoka bash wymaga w tym miejscu użycia polecenia *exit*, które jest odpowiednikiem użytej przez nas opcji *Quit* (wyjście z programu).

Dodatkowo musimy wziąć pod uwagę monitorowanie sygnału *SIGINT*. Pojawia się on w chwili naciśnięcia klawiszy *CTRL+C* przez użytkownika. Gdyby nasze menu było zwykłym programem, zostałyby po prostu zamknięte i musielibyśmy uruchomić je ponownie. Program jednak jest teraz powłoką, która spowodowałaby w takim przypadku także wylogowanie użytkownika. A zanim to nastąpi, powinniśmy zamknąć wszystkie uruchomione programy i pliki oraz wyłączyć odtwarzaną muzykę. Powód, dla którego musimy zamykać

pliki podczas użytkowania plików zablokowanych, opisano w Ramce 2: Priorytet dostępu. Trzeba zatem ustawić odpowiednią pułapkę, która wykrywałaby naciśnięcie takiej kombinacji klawiszy:

```
/* Dodane do głównej funkcji */
signal(SIGINT, CloseHandler);

/* Ten parametr sprawia, że
uchwyt może być ponownie użyty
do obsługi różnych sygnałów */

void CloseHandler(int Signal)
{
    puts('Exiting...');
    /* Stop all music here */
    exit(0);
}
```

Zamiast obsługi sygnału *CTRL+C*, możemy go kompletnie zignorować. Poniżej podaliśmy skuteczne rozwiązanie:

```
signal(SIGINT, SIG_IGN); /* Zignoruj
identical to ignore */
(SIGINT); /*
```

Prawdopodobnie będziemy chcieli przechwytywać także inne sygnały. Gdybyśmy starali się stworzyć np. powłokę matematyczną, należałoby zająć się obsługą wyjątków zmiennoprzecinkowych (*SIGFPE*). Pełna lista sygnałów znajduje się w */usr/include/bits/signum.h*, nie wszystkie z nich można jednak wykrywać (np. *SIGKILL* czy *SIGSTOP*).

Jak kobieta

Dużo więcej kłopotu podczas programowania powłoki stwarza funkcja *system*, która może nie wierzyć lub nie, ładuje swój własny interpreter poleceń do wykonania podanego polecenia! Chodzi o powłokę *sh* z dowiązaniem symbolicznym do *bash*. Nie jest to problem sam w sobie (oszczędza czas na załadowanie innej powłoki), ale jeżeli tworzymy powłokę dla specyficznego środowiska, taka funkcja może okazać się problematyczna (będziemy wtedy musieli posiadać kopię *bash* na dysku).

Zamiast *system* użyjemy funkcji *vfork* w połączeniu z *execvp*. Pierwsza z nich rozdziela obecny program na dwa identyczne procesy. Pierwsza część rozwidlenia (zwana procesem macierzystym [ang. parent]) pracuje jak zwykła powłoka, a druga część (zwana procesem potomnym [ang. child]) wykonuje nasz program zewnętrzny przy pomocy drugiej funkcji

Listing 1. Samodzielne menu

```
#include <stdio.h>
#include <stdlib.h>

void PrintMenu(void)
{
    puts("Menu Options\n");
    puts("1. Play CD");
    puts("2. Stop CD");
    puts("3. Play MP3s");
    puts("4. Mute");
    puts("5. Unmute");
    puts("9. Quit");
}

int GetUserOption(void)
{
    char opt[16];

    printf("Option: ");
    fgets(opt, sizeof(opt), stdin);
    opt[sizeof(opt)-1] = '\0';
    return atoi(opt);
}

void ProcessMenu(int iOption)
{
    switch(iOption)
    {
        case 1:
            system("cdcd -d /dev/cdrom
play");
            break;
        case 2:
            system("cdcd -d /dev/cdrom
stop");
            break;
        case 3:
            system("mplayer mp3s/*.mp3
&>/dev/null");
            break;
        case 4:
            system("aumix -d /dev/mixer -
v0");
            break;
        case 5:
            system("aumix -d /dev/mixer -
v99");
            break;
    }
}

int main(int argc, char **argv)
{
    int iOpt;

    puts("Multimedia Shell - v0.0\n");
    do
    {
        PrintMenu();
        iOpt = GetUserOption();
        ProcessMenu(iOpt);
    }
    while(iOpt != 9);

    return 0;
}
```

Ramka 2. Priorytet dostępu

Mimo że uprawnienia dostępu zabezpieczają system przed niepowołanymi gośćmi korzystającymi z odtwarzacza CD czy karty dźwiękowej, możemy jeszcze bardziej ograniczyć użycie *mmshell* do pojedynczego przypadku. Dzięki temu zabezpieczymy się przed sytuacją, w której dwóch użytkowników będzie wydawało sprzeczne ze sobą polecenia, a powłoka może odmówić dostępu obu użytkownikom, w przeciwieństwie do *music*.

Możemy również utworzyć prosty plik *lock* w katalogu */home/music*. Plik ten jest potrzebny tylko jako wskaźnik zalogowania da-

nego użytkownika i zabrania *mmshell* dostępu do nowych poleceń czy dodatkowych logowań tego użytkownika. Plik *lock* powinien zostać usunięty po wylosowaniu użytkownika.

Dopiero tutaj widać konieczność występowania sygnałów obsługi (np. SIGINT). Jeżeli, przykładowo, użytkownik naciśnie kombinację klawiszy CTRL+C przy uruchomionym *mmshell*, a sygnał nie był obsługiwany, plik *lock* nie zostałby nigdy usunięty i nikt nie mógłby zalogować się ponownie jako użytkownik *music* do chwili ręcznego skasowania użytkownika *root*.

execvp. Rozwidlenie jest konieczne, ponieważ *execvp* zastępuje istniejący proces, uniemożliwiając nam dalsze działanie interpretera poleceń. Tak więc dzięki rozbiciu procesu na dwie części, jedna z nich działa dalej, a druga używa się podczas pracy. Popatrzmy na Listing 2.

W ten sposób otrzymujemy niejako w promocji kolejną cechę – możliwość poznania identyfikatora procesu (ID) uruchomionego programu. Możemy później wykorzystać ten identyfikator do usunięcia procesu (a więc wyciszenia muzyki) w chwili naciśnięcia przez użytkownika kombinacji klawiszy CTRL+C.

```
kill(process_id, SIGKILL);
```

execvp jest jedną z odmian funkcji *exec*()*. Wszystkie z nich opisano w *man exec*, więc zapraszamy do lektury.

Wywołanie naszej funkcji usuwającej proces można przeprowadzić następująco:

```
char *args[] = { "cdcd", 2
"play", NULL };
ForkAndExec("cdcd", args);
```

Tutaj musimy nadać nazwę programowi w dwóch miejscach: nazwa pliku wykonywalnego i pierwszy argument. Przyjmuje się, że argument zerowy oznacza nazwę pliku. Nie musi to być prawdziwa nazwa pliku (jako że ktoś może korzystać z dowiązań symbolicznych), ale użycie prawdziwej nazwy nie będzie stanowić problemu.

Będąc programistami języka C, przyzwyczajaliśmy się już, że nazwa programu określana jest przez *argv[0]*. Nazwa pliku nie wymaga pełnej ścieżki dostępu, gdyż nasza wersja *execvp* przeszuka automatycznie katalogi */bin* i */usr/bin* (jeżeli nie podano ścieżki dostępu).

ni podczas programowania w powłoce, jest po prostu nieobecna, gdy pracujemy bez niej. Rozwiązanie przynoszą nam projektanci systemu *libc*. W rzeczywistości proponują oni dwa różne rozwiązania.

Pierwsze z nich polega na zamianie funkcji *execvp* na funkcję *execle*. Obsługuje ona zwykłe parametry programu w innym formacie i wstawia przed nimi pustą tablicę zmiennych środowiskowych.

```
execle("/usr/bin/cdcd", 2
"cdcd", "play", NULL, 2
ppEnviroVars);
/* Note: full path required */
```

Beze mnie...

Po usunięciu wywołania *system* z naszego interpretera poleceń straciliśmy więcej niż niepotrzebne wywołanie *bash*. Straciliśmy przyjaciela! Wszystko, czego dostarcza nam powłoka, zostało stracone (Ramka 1: Niektóre cechy powłoki odświeży nam pamięć), np. przekierowanie (ukrywa nadmierne dane wyjściowe *mplayer*), zmienne środowiskowe (w szczególności polecenie *PATH*), czy symbole wieloznaczne. Utrata ostatniej funkcji oznacza, że nie będziemy mogli już korzystać z **.mp3* w naszej liście odtwarzania. Próbując użyć tej funkcji, otrzymamy jedynie komunikat błędu mówiący o braku pliku **.mp3*. Problem ten możemy rozwiązać w trójnasób – podając nazwę każdego pliku w katalogu, wykorzystując zestaw poleceń *opendir-readdir-closedir* lub też powracając do polecenia *system*. Niestety, nie mamy tyle miejsca na łamach naszego pisma, aby zamieścić tutaj pełne rozwiązanie. Umieściliśmy je jednak na stronie internetowej magazynu *Linux Magazine* pod adresem <http://www.linux-magazine.pl/issue/02/mmshell2.c>

Kolejną funkcją powłoki, której brak będziemy mocno odczuwać, jest środowisko. Każda zmienna, od której będziemy uzależnie-

Drugie rozwiązanie polega na wykorzystaniu zmiennej globalnej! Zmienna nosi nazwę *environ* i ma taki sam format, jak opisana powyżej *ppEnviroVars*. Używana jest w pozostałych formach *non-execle*, funkcji *exec*()*.

Na koniec powinniśmy powiedzieć kilka słów o ścieżce wyszukiwania. Wiemy już jak działa *execvp*, przeszukując najpopularniejsze katalogi w poszukiwaniu plików wykonywalnych – *execplp* działa tak samo. Inne odmiany nie posiadają już jednak tej funkcji. Z reguły nie stwarza to wielkiego problemu, gdyż rozsądnie jest umieszczać pełne ścieżki dostępu do plików, które chcemy uruchamiać – zwiększa to bezpieczeństwo systemu.

Kapitan Huśtawka

W zależności od punktu widzenia, bezpieczeństwo jest fantastycznym wyzwaniem i miejscem wykorzystania nieskończonych możliwości lub też największym problemem administracyjnym dzisiejszych czasów. Cokolwiek byśmy nie myśleli, bezpieczeństwo jest na pewno najbardziej istotne. Nawet programiści nie są odporni na zwracanie uwagi na bezpieczeństwo. W końcu to właśnie przede wszystkim programiści popełniają błędy w zabezpie-

Listing 2. Dzielenie na dwa procesy

```
void ForkAndExec(const char *pName,
const char **ppArgs)
{
pid_t process_id;
/* Używamy vfork zamiast fork,
ponieważ tak jest szybciej */
/*jeśli zamierzamy jedynie, aby
potomek wywoływał execvp */
process_id = vfork();
switch(process_id)
{
case -1:
printf("ERROR! Could not spawn
%s...", pName);
return;
case 0:
execvp(pName, ppArgs);
exit(0);
default:
/* Proces macierzysty */
printf("Spawned PID %d\n",
process_id);
}
return process_id;
```

zeniach programów! Rozważmy jednak, czym jest błąd zabezpieczeń?

Szybki rzut oka na publikowane rady (typu BugTraq) czy też wiadomości z zakresu (nie)bezpieczeństwa w Linux-Magazine, pozwala wyróżnić kilka głównych problemów związanych z ochroną systemu. Najczęściej problem jest następujący:

Wersja X programu Y posiada błąd przepełnienia bufora, który może być wykorzystany, umożliwiając wykonywanie dowolnych poleceń przez intruza podszywającego się pod bieżącego użytkownika.

Mimo że jest to poważny problem dla serwerów głównych (root) lub mających dostęp do grupy wheel, dla nas może być to sprawa marginalna, nie wymagająca naszej uwagi. W końcu jest to tylko prosty interpreter poleceń. Jeżeli mamy go używać, musimy jednak posiadać odpowiednie uprawnienia do pracy na tym komputerze (moglibyśmy wyrazić więcej szkód używając bash i polecenia *rm!*). Prawda? Każdy problem związany z bezpieczeństwem jest istotny. Jeżeli korzystamy z *mmshell* w ograniczonym środowisku (np. do obsługi kiosku internetowego czy bankomatu), użytkownicy nie będą mieli możliwości zmiany powłoki. Luka w bezpieczeństwie może to umożliwiać, przez co hakerzy znajdują punkt wejścia do ataku.

Bezpieczeństwo powłoki można osiągnąć solidnymi technikami programistycznymi: kompilując kod należy brać pod uwagę każde ostrzeżenie, które kompilator nam podaje, można też skorzystając z narzędzi typu Splint [1] czy Flawfinder [2] do przeprowadzenia szczegółowych testów. Powinniśmy sprawdzić granice bezpieczeństwa (najlepiej nie korzystając z narzędzi typu *sprintf* i *strcpy*), potwierdzić wejście użytkownika (kontrola, usuwanie i sekwencje wyjścia) oraz unikać niepewnego oprogramowania. Ostatni punkt ciężko zrealizować, gdyż będziemy uruchamiać inne programy, a kontrolę już przekazaliśmy naszej powłoce. Tworzenie bezpiecznych systemów staje się coraz trudniejsze z powodu wielkiej ilości nieznanego oprogramowania (np. małych programów typu klient i edytorów).

Gdy już znaleźliśmy (lub napisaliśmy) bezpieczny program, możemy go uruchomić. Aby utrzymać odpowiedni poziom bezpieczeństwa, należy uruchomić program z pełną ścieżką dostępu. Nie będzie w ten sposób miejsca na swobodną interpretację położenia np. programu 'mplayer', zamiast którego zostanie uruchomiony (pomyłkowo lub celowo) koń trojański. Oczywiście program wykonywalny powinien być przechowywany w bezpiecznym katalogu, do którego dostęp ma wyłącznie użytkownik

główny. Aby potwierdzić ścieżkę dostępu do pliku wykonywalnego, spróbujmy wpisać:

```
$ which mplayer
/usr/local/bin/mplayer
```

Bezpieczne programowanie jest tematem tyle rozległym, co skomplikowanym. Napisano na ten temat wiele książek i opracowań. Niektóre z nich są dostępne pod adresami [3] i [4].

Śpiąc w samochodzie

Po skończeniu projektu powinniśmy go przetestować. Stwórzmy zatem użytkownika o nazwie *music* i dołączmy go do grupy *audio*.

```
# adduser --ingroup audio music
```

Upewnijmy się jeszcze, że grupa *audio* posiada odpowiednie uprawnienia dostępu do sterowania odtwarzaczem CD oraz karty muzycznej i będziemy już gotowi do testów funkcjonalnych użytkownika. Jako że utworzony przez nas interpreter poleceń jest bardzo restrykcyjny, nie uruchamiamy go na naszym koncie użytkownika, aby później nie błądzić po omacku i być w rezultacie zmuszonym do zalogowania się jako użytkownik główny (root), żeby usunąć powstałe usterki.

Testowanie *mmshell* to proces dwuetapowy. W pierwszym musimy zalogować się jako użytkownik *music* z powłoką i korzystać z programu, jak ze zwykłego systemu menu. Potwierdzi to posiadanie ustalonych wcześniej właściwych uprawnień i działanie sygnałów. W drugim etapie musimy skonfigurować użytkownika *music* w taki sposób, aby po zalogowaniu ładowany był nasz nowy interpreter poleceń.

Mimo że system Linux umożliwia zwykłemu użytkownikowi zmianę powłoki, nie pozwala jednak na dodanie własnych powłok do systemu. Jedynymi dozwolonymi powłokami są te, znajdujące się w pliku */etc/shells* file. Dzięki temu zapewniamy bezpieczeństwo i stabilność systemu. Użytkownicy wierzą, że ich interpreter poleceń będzie dyskretny, nie zapisując każdego naciśnięcia klawisza w pliku dziennika. Nawet gdyby użytkownik zalogował się do systemu używając bezpiecznej powłoki (takiej jak ssh), naciśnięcia klawiszy mogą być nadal przechwytywane przez interpreter poleceń podobny do konia trojańskiego (w chwili, gdy powłoka ssh przekaże sterowanie domyślnej powłoce użytkownika). Na szczęście tylko użytkownik główny (root) ma przywileje umożliwiające zmianę powłok, więc ryzyko zarażenia się trojanem jest stosunkowo niskie.

Po wnikliwej analizie nowego kodu po-

włoki (jako że napisaliśmy ją sami, wiemy, że jest stosunkowo bezpieczna!), kopiujemy ją do katalogu */usr/bin* i dodajemy jej nazwę do listy dostępnych interpreterów poleceń. Pamiętajmy, żeby podczas wykonywania tej operacji posiadać przywileje użytkownika głównego (roota).

```
# cp mmshell /usr/bin
# echo /usr/bin/mmshell >>/etc/
/shells
```

Teraz musimy sprawić, aby znak zachęty do logowania używał naszej powłoki zamiast powłoki bash. Zmian możemy dokonać bezpośrednio w pliku */etc/passwd*, jednak nie zalecamy takiego rozwiązania. Lepiej użyć programu *chsh*. Działa on na dowolnym koncie użytkownika, ale zmiana powłoki także innych użytkowników wymaga użycia konta użytkownika głównego.

```
$ chsh -s /usr/bin/mmshell
```

Zmiana powłoki będzie obowiązywać od chwili ponownego zalogowania do systemu. Tak więc napiszmy „exit” i zalogujemy się ponownie. Jeżeli wszystko działa prawidłowo (a nie ma powodu, żeby nie działało poprawnie), być może będziemy musieli zmienić ścieżki dostępu dla poszczególnych programów, np. *aumix* czy *mplayer*), zobaczymy na ekranie:

```
mymachine login: music
Password:
Linux tori 2.4.19 #44 SMP Sun
Dec 28 19:07:54 GMT 2003 i686 >
unknown
Last login: Sun Jan 4 >
14:32:26 2004 from mymachine

Powłoka multimedialna >
- wersja 0.1

Opcje Menu

1. Odtwarzaj CD
2. Zatrzymaj CD
3. Odtwarzaj pliki MP3
4. Wycisz
5. Włącz dźwięk
9. Wyjście
```

Oto nasze menu! Powłoka ma podstawową funkcjonalność, ale nic nie stoi na przeszkodzie, żeby zaktualizować *mmshell*, dodając nowe funkcje zgodnie z potrzebami. Można także napisać nowy interpreter poleceń. ■